Computing 39, 327 – 344 (1987)

Computing
© by Springer-Verlag 1987

A Cutting Plane Algorithm for Minimum Perfect 2-Matchings

M. Grötschel, Augsburg, and O. Holland, Bonn

Received January 13, 1987

Abstract — Zusammenfassung

A Cutting Plane Algorithm for Minimum Perfect 2-Matchings. We describe an implementation of a cutting plane algorithm for the minimum weight perfect 2-matching problem. This algorithm is based on Edmonds' complete description of the perfect 2-matching polytope and uses the simplex algorithm for solving the LP-relaxations coming up. Cutting planes are determined by fast heuristics, or, if these fail, by an efficient implementation of the Padberg-Rao procedure, specialized for 2-matching constraints. With this algorithm 2-matching problems on complete graphs with up to 1000 nodes (i.e., 499.500 variables) have been solved in less than 1 hour CPU time on a medium speed computer.

AMS Subject Classifications: 05-04, 05C45, 90C10.

Key words: Matching, cutting plane algorithm, polyhedral combinatorics.

Ein Schnittebenenverfahren für minimale perfekte 2-Matchings. Wir beschreiben die Implementierung eines Schnittebenenverfahrens zur Bestimmung minimaler gewichteter perfekter 2-Matchings. Der Algorithmus baut auf der vollständigen Beschreibung des perfekten 2-Matching-Polytops, die Edmonds angegeben hat, auf und verwendet die Simplexmethode zur Lösung der im Verfahren auftretenden LP-Relaxierungen. Schnittebenen werden entweder mit schnellen Heuristiken bestimmt, oder, falls diese nicht erfolgreich sind, mit einer effizienten und auf 2-Matching-Ungleichungen abgestimmten Implementierung des Padberg-Rao-Verfahrens. Mit diesem Algorithmus konnten 2-Matching-Probleme in vollständigen Graphen mit bis zu 1000 Knoten, d. h. mit bis zu 499.500 Variablen, in weniger als einer Stunde CPU-Zeit auf einem Rechner mittlerer Leistung gelöst werden.

1. Introduction

Just as matching theory is one of the central topics of graph theory, the development of efficient matching algorithms has been a major issue in combinatorial optimization and computer science – see [9] for the state-of-the-art and the history of this subject. Algorithmic research has mainly concentrated on the design of fast methods for the perfect 1-matching problem – to which, in fact, many of the other types of matching problems can be reduced. Usually, however, these reductions enlarge the size of a problem significantly, and it is much more appropriate to design special purpose algorithms if other matching problems are of interest.

The problem we are aiming at is the weighted minimum perfect 2-matching problem which we call just *2-matching problem* henceforth. An instance of this problem is specified by a graph $G=(V,E)$ (we assume, without loss of generality that G has no loops and no multiple edges) and edge weights $c_e \in \mathbb{R}$ for all $e \in E$. We want to find a perfect 2-matching M in G with minimum total weight $c(M) := \sum_{e \in M} c_e$. A perfect 2-matching (henceforth just 2-*matching*) is an edge set M such that each node is contained in exactly two edges of M.

In his seminal paper [2] Edmonds proved that the 2-matching problem can be solved in polynomial time. He also gave a complete description of the 2-matching polytope – see Section 2. However, implementing Edmonds' algorithm efficiently is a nontrivial task. The only working implementation we are aware of is the code of Edmonds, Johnson and Lockhart described in [3]. But this code can handle only rather small graphs of no more than 1500 edges. We will describe an implementation of a cutting plane algorithm for this problem. Our code is theoretically non-polynominal – it could be made polynominal, though – but empirically quite efficient. It has successfully solved 2-matching problems on graphs with half a million edges.

The main interest in the 2-matching problem stems from the fact that it is a nice relaxation of the symmetric travelling salesman problem. This applies to our case also. We have designed the algorithm described here mainly to solve relaxations of the TSP and to get good lower bounds. This code will be merged with another code for the 1-matching problem – see [7] – modified and enhanced by further "tricks" to solve large travelling salesman problems. We found out empirically, however, that 2-matching and travelling salesman problems have to be handled differently. This means that the most efficient versions of our codes for the 2-matching problem and the TSP use different strategies for recognizing 2-matching constraints, and so there are significant differences between these algorithms.

The remaining part of this paper is organized as follows. In Section 2 we outline the theoretical background (polyhedral combinatorics, separation etc.) of our algorithm. A description of our version of the Padberg-Rao procedure for finding 2-matching constraints is given in Section 3. The complete cutting plane algorithm is stated in Section 4 and computational results are reported in Section 5.

2. The Polyhedral Approach to the 2-Matching Problem

We will now describe the polyhedral background of our approach.

Let a graph $G = (V, E)$ be given. For an edge set $F \subseteq E$, the vector $\chi^F = (\chi^F_e)_{e \in E}$ with $\chi^F_e = 1$ if $e \in F$ and $\chi^F_e = 0$ if $e \in E \setminus F$ is called the *incidence vector of* F. The 2-*matching polytope* $Q(G)$ of the graph G is the convex hull of all incidence vectors of 2-matchings of G, i.e.,

$$Q(G) = \text{conv}\{\chi^M \in \mathbb{R}^E \mid M \subseteq E \text{ 2-matching}\}.$$

A major result of [2] is the complete description of $Q(G)$ by linear inequalities and equations. To state this, let us introduce the following notation.

For $W \subseteq V$, the symbol $E(W)$ denotes the set of all edges of G with both endnodes in W, while $\delta(W)$ denotes the set of edges of G with one endnode in W and the other in

$V \setminus W$. $\delta(W)$ is called a *cut*. For any $x \in \mathbb{R}^E$ and any $F \subseteq E$, the sum $\sum_{e \in F} x_e$ is abbreviated by $x(F)$. Recall that an inequality $a^T x \leq \alpha$ is called *valid* for a polytope P if $P \subseteq \{x \mid a^T x \leq \alpha\}$ and that, for a valid inequality $a^T x \leq \alpha$, $F := \{x \in P \mid a^T x = \alpha\}$ is called a *face* of P. A face of P is a *facet* if $\dim(F) = \dim(P) - 1$, where $\dim(P)$ is the maximum number of affinely independent points in P minus 1.

Since $Q(G)$ is contained in the unit hypercube the inequalities

(2.1) $$0 \leq x_e \leq 1 \quad \text{for all } e \in E$$

are valid for $Q(G)$. As every node of G is contained in exactly 2 edges of a 2-matching, every point in $Q(G)$ satisfies the equations

(2.2) $$x(\delta(v)) = 2 \quad \text{for all } v \in V.$$

Observe that every 2-matching is the disjoint union of circuits. And since every circuit intersects every cut in an even number of edges, every 2-matching intersects every cut in an even number of edges. This graph-theoretical observation implies that every point in $Q(G)$ satisfies the following system of inequalities

(2.3) $$x(T) - x(\delta(W) \setminus T) \leq |T| - 1 \quad \text{for all } W \subseteq V \text{ and all } T \subseteq \delta(W) \text{ with } |T| \text{ odd}.$$

By adding, to any of the inequalities of (2.3), the equation $\sum_{v \in W} x(\delta(v)) = 2|W|$, we see that the system (2.3) is equivalent to

(2.4) $$x(E(W)) + x(T) \leq |W| + \frac{|T| - 1}{2} \quad \text{for all } W \subseteq V \text{ and all } T \subseteq \delta(W) \text{ with } |T| \text{ odd}.$$

Using, for each vector $x \in \mathbb{R}^E$ with $0 \leq x \leq 1$, the vector $\bar{x} \in \mathbb{R}^E$ defined by $\bar{x}_e := 1 - x_e$, one immediately notices that the system (2.3) is also equivalent to

(2.5) $$\bar{x}(T) + x(\delta(W) \setminus T) \geq 1 \quad \text{for all } W \subseteq V \text{ and all } T \subseteq \delta(W) \text{ with } |T| \text{ odd}.$$

The inequalities (2.3), (2.4), or (2.5) are usually called *2-matching constraints*. Edmonds [2] proved, that the inequalities (2.1), (2.2), and the 2-matching constraints suffice to describe $Q(G)$, i.e.:

(2.6) Theorem: *For every graph $G = (V, E)$,*

$$Q(G) = \{x \in \mathbb{R}^E \mid x \text{ satisfies (2.1), (2.2), and (2.3)}\}.$$

As mentioned above, the system (2.3) can be replaced either by the inequality system (2.4) or by (2.5). For various reasons, it is important to consider only inequalities that define facets of $Q(W)$. For the complete graph K_n on n nodes, the case we will be concerned with later, the facets of $Q(K_n)$ have been characterized in [6]. From this the following description of $Q(K_n)$ can be derived.

(2.7) Theorem: *The following system of inequalities and equations is a complete and nonredundant characterization of $Q(K_n)$, $n \geq 5$:*

(*i*) $0 \leq x_e \leq 1$ *for all* $e \in E$, (*ii*) $x(\delta(v))=2$ *for all* $v\in V$,

(*iii*) $x(T)-x(\delta(W)\setminus T)\leq|T|-1$ *for all* $W\subseteq V$ *and all* $T\subseteq\delta(W)$ *such that*

(*a*) *T is a matching*,

(*b*) $|T|\geq 3$ *and* $|T|$ *odd, or* $|T|=1$ *and* $4\leq|W|\leq n-4$

(*c*) $W\in\mathscr{V}$, *where* $\mathscr{V}$ *is any set of subsets of V such that*

$$W\in\mathscr{V}\Leftrightarrow V\setminus W\notin\mathscr{V}.$$

These results imply that the 2-matching problem can be solved by solving the linear program

$$\text{(2.8)} \qquad \min c^T x, x \text{ satisfies (2.1), (2.2), and (2.3).}$$

Instead of the system (2.3), the system (2.4), or (2.5), or the one described in (iii) of (2.7) can be used in (2.8). The number of constraints of (2.8) is exponential in the encoding length of the graph G, but this is not too important since one can derive from the ellipsoid method – see [8] – that the linear program (2.8) can be solved in polynomial time if and only if the following separation problem can be solved in polynomial time.

(2.9) Separation Problem for *$Q(G)$: Given a vector $y\in\mathbb{Q}^E$, decide whether $y\in Q(G)$ and if y is not in $Q(G)$, find a hyperplane separating y from $Q(G)$.*

Since $Q(G)$ is defined by, e.g., the inequalities (2.1), (2.2), and (2.3), the separation problem for $Q(G)$ can be solved by checking whether a given vector $y\in\mathbb{Q}^E$ satisfies (2.1), (2.2), and (2.3). By substituting y into (2.1) and (2.2) it is of course trivial to check these inequalities. It is not obvious to see whether y satisfies all inequalities (2.3). But Padberg and Rao [10] found an algorithm – to be outlined in the next section – with which this can be done in polynominal time.

A combination of this separation algorithm with the ellipsoid method results in a polynominal time solution procedure for the 2-matching problem. This procedure, however, is quite inefficient in practice.

We have replaced the ellipsoid method by the simplex method, added a few heuristics, designed special data structures and search techniques etc. to make it work in practice. These modifications will be described in the next two sections.

3. The Separation Subroutine

We will now outline the algorithms we have implemented to solve the separation problem for the system (2.1), (2.2), (2.3). In fact, the initial linear program we solve consists of the bounds (2.1) and the equations (2.2). So these inequalities and equations will be satisfied by any point to be considered. Checking the system (2.3) is the crucial point.

As remarked before, we have implemented our algorithm for complete graphs K_n since all our applications are 2-matching problems on complete graphs. We will describe later how arbitrary graphs can be handled and restrict our attention now to the complete case. Define, for the complete graph $K_n=(V,E)$,

$$\text{(3.1)} \qquad Q_1^n := \{x\in\mathbb{R}^E \mid x \text{ satisfies (2.1) and (2.2)}\}.$$

It is easy to see that

$$Q(K_n) = \operatorname{conv}\{x \in Q_1^n \mid x \text{ integral}\},$$

so

$$\text{(3.2)} \qquad \max c^T x, \; x \in Q_1^n, \; x \text{ integral}$$

is an integer programming formulation of the 2-matching problem. The vertices of Q_1^n are either incidence vectors of 2-matchings or fractional points with nice structural properties. Namely, all fractional components of a vertex y of Q_1^n have value $\frac{1}{2}$, and moreover, the set of edges $F_y := \{e \in E \mid y_e = \frac{1}{2}\}$ is the disjoint union of an even number of circuits of odd length. This fact in mind, we have designed a heuristic that searches for such structures and may produce violated inequalities of type (2.3). It is guaranteed to find, for each fractional vertex y of Q_1^n, a 2-matching constraint violated by y; but it frequently also finds cutting planes if vertices of other (tighter) LP-relaxations are given to it.

(3.3) A Separation Heuristic for the Inequalities (2.3):
Input: A point $y \in Q^E$ satisfying (2.1) and (2.2) and a rational number $0 < \varepsilon < \frac{1}{2}$.

1. Construct the graph $G_\varepsilon := (V, E_\varepsilon)$ defined by $E_\varepsilon := \{e \in E \mid \varepsilon \le y_e \le 1 - \varepsilon\}$.
2. Determine the connected components $(V_1, E_1), \dots, (V_k, E_k)$ of G_ε with $|V_i| \ge 3$, $i = 1, \dots k$.
3. FOR $i = 1, \dots, k$ DO the following:
   a) Sort the edges $e \in E$ with $e \in \delta(V_i)$ and $y_e > 1 - \varepsilon$ in nonincreasing order to obtain a sequence $e_1, \dots, e_p$ of edges. (If p is even, then add a largest edge e with $y_e < \varepsilon$.)
   b) Set $z := y(E(V_i)) + y_{e_1}$ and $j := 1$.
   c) WHILE $j \le p$ DO;

```
        IF $z > |V_i| + (j-1)/2$, THEN
        DO;
            $T_i := \{e_1, \dots, e_j\}$;
            STORE $(V_i, T_i)$;
            LEAVE while-loop;
        END;
        SET $z := z + y_{e_{j+1}} + y_{e_{j+2}}$;
        SET $j := j + 2$;
    END WHILE;
```

□

All pairs (V_i, T_i) stored during the execution of (3.3) define inequalities

$$x(E(V_i)) + x(T_i) \le |V_i| + (|T_i| - 1)/2$$

that are violated by y. By running experiments with fractional solutions that came up in our algorithm we found that $\varepsilon := 0.3$ is a good choice for the heuristic described in (3.3). The worst-case running time of (3.3) is $O(n^2)$; for our choice of ε and the inputs supplied by the cutting plane algorithm, it is empirically linear on the average.

All violated inequalities found in heuristic (3.3) will be added to the current LP. Typically, in the first iterations of the cutting plane procedure, heuristic (3.3) is quite effective and finds many cutting planes. If it fails to find any violated inequality, the Padberg-Rao procedure, that provably will produce such an inequality, if any exists, is called. We outline this method now.

The first step in this method is a procedure that replaces every edge by two edges in series and labels nodes in a certain way.

(3.4) Labeling Procedure:
Input: A graph $\hat{G}=(\hat{V},\hat{E})$ with edge weights $\hat{y}_e\in\mathbb{R}$ such that $0\leq\hat{y}_e\leq 1$ and a set $\hat{T}\subseteq\hat{V}$, $|\hat{T}|$ even.

Output: A graph $G^*=(V^*,E^*)$ with edge weights z_e, $0\leq z_e\leq 1$, and a set $T^*\subseteq V^*$, $|T^*|$ even.

```
1. Set $V^*:=E^*:=\emptyset$, $T^*:=\hat{T}$.
2. Order the edges in $\hat{E}$ arbitrarily, say $e_1,\ldots,e_m$.
3. FOR $k=1$ TO $m$ DO;
     Suppose $e_k=uv$, set $V^*:=V^*\cup\{u,v,i_{e_k}\}$ (where $i_{e_k}$ is a new node representing
     the edge $e_k$), $E^*:=E^*\cup\{ui_{e_k},vi_{e_k}\}$ (so $uv$ is replaced by two edges in series),
     $T^*:=T^*\,\Delta\,\{u,i_{e_k}\}$ (where $\Delta$ denotes the symmetric difference), and set
          $z_{ui_{e_k}}:=1-\hat{y}_{e_k}$, $z_{vi_{e_k}}:=\hat{y}_{e_k}$.
   END                                                                    □
```

It is convenient to call the nodes in the final set T^* the nodes labeled *odd* and to call the nodes in $V^*\setminus T^*$ labeled *even*. A cut $\delta(W^*)$, $W^*\subseteq V^*$, in G^* is called *odd* iff $|T^*\cap W^*|$ is odd.

Let, as before, $K_n=(V,E)$ be the complete graph on n nodes and assume that we have a point $y\in\mathbb{Q}^E$ that satisfies (2.1) and (2.2). We first construct the graph $G_y=(V,E_y)$, where

$$E_y:=\{e\in E\mid y_e>0\}.$$

Now the Padberg-Rao procedure calls the labeling method described in (3.4) with input $G_y=(V,E_y)$, weights y_e, $e\in E_y$, and $\hat{T}:=\emptyset$ to construct the new graph $G^*=(V^*,E^*)$ with weights z_e, $e\in E^*$, and an even set $T^*\subseteq V^*$ of odd nodes.

For each odd cut $\delta(W^*)$ of G^* we can construct a corresponding cut in G_y by setting

$$W:=W^*\cap V$$

and

$$T:=\{uv\in\delta(W)\mid ui_e\in\delta(W^*)\text{ with } z_{ui_e}\text{ has been set to } 1-y_e,\text{ where } e=uv\}.$$

Note that $|T|$ is odd since $\delta(W^*)$ is an odd cut in G^*. By definition

$$z(\delta(W^*))=y(\delta(W)\setminus T)+\bar{y}(T),$$

and so, $\delta(W^*)$ is an odd cut of z-weight less than 1 if and only if the corresponding inequality $\bar{x}(T)+x(\delta(W)\setminus T)\geq 1$ of type (2.5) is violated by y.

It follows that y satisfies all inequalities (2.5) (or equivalently, all inequalities (2.4) or (2.3)) if and only if the minimum weight of an odd cut in G^* is at least 1.

Padberg and Rao [10] have shown that an odd cut of minimum weight can be determined by a modification of the Gomory-Hu procedure – see [5] or [9]. This algorithm is polynomial but requires $O((|V|+|E_y|)^4)$ operations in the worst-case. To get a practically efficient separation routine it is vital to reduce this time bound considerably for the type of problems that arise in this context. This can be done by various preprocessing procedures and modifications of the standard Gomory-Hu algorithm. We will briefly explain a few that have resulted in significant empirical running time improvements.

The Padberg-Rao procedure continues as follows. Starting with $G^*=(V^*,E^*)$ and an even set $T^*\subseteq V^*$ a Gomory-Hu tree (or flow-equivalent-tree) H is build up recursively in the following way.

Initially, the tree H consists of one tree-node, representing all nodes V^* of G^*, and no tree-edge. We call a tree-node *exhausted* if it represents a node set $W^*\subseteq V^*$ with $|T^*\cap W^*|=1$. If all tree-nodes are exhausted the algorithm stops. Otherwise we pick an unexhausted tree-node w^*. This node represents some set $W^*\subseteq V^*$. Let $W_1,\ldots,W_k$ denote the node sets of the components of $H-w^*$ and let, for $i=1,\ldots,k$, W_i^* be the union of the node sets in G^* represented by the nodes in W_i. Construct a graph G_w^* from G^* by shrinking successively the nodes sets $W_1^*,\ldots,W_k^*$ into nodes $w_1^*,\ldots,w_k^*$, say, and modify the edge weights accordingly. Then we choose two nodes $u,v\in T^*\cap W^*$ and determine a minimum weight cut $\delta(U^*)$ in G_w^* separating u and v. The tree-node w^* is now replaced by two new nodes u^*, representing $U^*\cap W^*$, and v^*, representing the nodes $\{x\in W^*\mid x\notin U^*\}$. The Gomory-Hu-tree is updated in a certain way and the process is repeated until all tree-nodes are exhausted.

Padberg and Rao have shown that every tree-edge with weight less than 1 in the final Gomory-Hu-tree that separates the tree into two odd components yields a cut $\delta(W)$ and an odd subset T of $\delta(W)$ such that the corresponding inequality (2.5) is violated by y. If no such tree-edge exists, all these inequalities are satisfied by y.

In a former version of our algorithm we stopped constructing the Gomory-Hu-tree as soon as a tree-edge of the present Gomory-Hu-tree had been determined whose weight was less than 1 and separated the present tree into two odd components. The corresponding cutting plane was added to the current LP and no attempt to find further violated inequalities was made. By comparing this procedure with the strategy to build up the whole Gomory-Hu-tree and adding all cutting planes provided by the tree we found that the latter strategy is superior with respect to overall running time – see Section 5. So this strategy is part of our present code.

Let us now discuss four features that contributed to considerable speed-ups of the Padberg-Rao algorithm.

Note first, that to build up the whole Gomory-Hu-tree, a max-flow algorithm has to be called $|T^*|-1$ times. In the worst-case $|T^*|=|V|+|E_y|$. In general $|E_y|$ may be $O(|V|^2)$, but in our application $|E_y|\le c|V|$ holds empirically for a rather small constant c. Moreover, by preprocessing and applying the labeling method in a special way, we can get

$$|T^*| \leq |\{e \in E_y \mid 0 < y_e < 1\}| + \kappa(G').$$

where $\kappa(G')$ is the number of components of G' defined below. This is best possible.

The preprocessing consists simply of removing all edges e from E_y with $y_e = 1$, labeling the endnodes of such edges, and deleting those nodes from G_y that are endnodes of two edges with y-weight equal to 1. More exactly, we set $V' := \{v \in V \mid v$ is incident to at least one edge e with $0 < y_e < 1\}$, $E' := \{e \in E_y \mid 0 < y_e < 1\}$, $G' := (V', E')$ and $T' := \{v \in V' \mid v$ is incident to exactly one e with $y_e = 1\}$. Now we call the labeling procedure (3.4) with input $G' = (V', E')$, y'(which is the vector y restricted to E'), and T' to obtain a graph $G^* = (V^*, E^*)$ with edge weights z and an even set $T^* \subseteq V^*$. It is easy to see that a Gomory-Hu tree obtained from this graph has a tree edge of weight less than 1 separating the Gomory-Hu tree into two odd components if and only if a Gomory-Hu tree derived from the graph G^* obtained by the labeling method (3.4) with input G_y, y, and $\hat{T} = \emptyset$ has such an edge.

Moreover, instead of ordering the edges of $\hat{E}$ in step 2 of (3.4) arbitrarily, we process them using depth-first search as follows. We start depth-first search with an arbitrary node $v_0 \in \hat{V}$. A node u is processed after all its successors (in the depth-first search tree) have been processed. Let q be the number of edges in $\delta_{\hat{G}}(u)$ that are not edges of the DFS-tree, let $l = 1$ if $u \in T^*$ and $l = 0$ otherwise, and let p be the predecessor of u. First, for all edges $e = uv \in \delta_{\hat{G}}(u)$ not in the DFS-tree we set $z_{ui_e} := 1 - \hat{y}_e$, $z_{vi_e} := \hat{y}_e$, and $T^* := T^* \cup \{i_e\}$. Then, for $e = up$, if $q + l$ is even we set $z_{ui_e} := \hat{y}_e$, $z_{pi_e} := 1 - \hat{y}_e$, $T^* := (T^* \setminus \{u\}) \Delta \{i_e, p\}$, and if $q + l$ is odd we set $z_{ui_e} := 1 - \hat{y}_e$, $z_{pi_e} = \hat{y}_e$, $T^* := (T^* \setminus \{u\}) \cup \{i_e\}$. Obviously, for each component (V_i', E_i') of G' we have $|T' \cap V_i'| < |E_i'|$, and it is easy to see that this labeling procedure stops with $|T^* \cap V_i'| = |E_i'|$ if $|E_i'| + |T' \cap V_i'|$ is even and with $|T^* \cap V_i'| = |E_i'| + 1$ if $|E_i'| + |T' \cap V_i'|$ is odd. Clearly, this is the best possible labeling that can be achieved.

The performance of the max-flow algorithm (which is called $|T^*| - 1$ times in the Gomory-Hu procedure) is quite crucial. We have compared various max-flow algorithms on the type of graphs that come up in this application. Our own implementation of the variant of the primal simplex algorithm described in [4] was the best method with respect to empirically observed running times.

Recall from the outline of the Gomory-Hu procedure that the graphs, to which the max-flow algorithm is applied, are constructed from G^* (and T^*) by shrinking certain node sets that can be read from the current Gomory-Hu tree. As mentioned above, we can replace the graph G^* constructed from G_y by a (usually) much smaller graph, and also the size of T^* can be controlled efficiently. We observed in a former version of our algorithm that the shrinking procedure, if programmed in a straightforward way, is rather time consuming. In our present version we proceed, roughly, as follows. We do not shrink from scratch every time. After having determined a minimum weight cut in some current graph $\hat{G} = (\hat{V}, \hat{E})$ we immediately construct the (at most) two possible new graphs that can be derived from the new Gomory-Hu tree by shrinking certain node sets in $\hat{G}$ and store these two graphs for future processing on a stack and remove $\hat{G}$ from the stack. To save storage space we place the graph with the smallest number of nodes in T^* (but at least two) on top of the stack.

Moreover, by carefully analysing the special structure of the graphs that come up, one can see that the whole subdivision procedure (replacing an edge uv by the two edges ui_e and vi_e in series) does not have to be performed explicitly for the max-flow calculation. The max-flow algorithm can be run on graphs obtained from the original graph G_y, even better from G', by shrinking certain node sets. This is due to the fact that a minimum cut that separates two nodes u and v will always contain the edge ui_e or vi_e with the smaller z-value. The only edges that have to be subdivided explicitly are those incident to the two nodes s, t in T^* for which a minimum weight $[s, t]$-cut is to be determined. Since T^* contains only new nodes i_e of degree two, with at most one possible exception, this shows that all max-flow calculations can be performed essentially on minors of G_y resp. G'. This observation and its (tedious) implementation resulted in considerable running time improvements.

The outline of our version of the Padberg-Rao resp. Gomory-Hu procedure may seem rather superficial. We have tried to explain the basic ideas. The technical details, data structures etc. are quite involved and their exact description is beyond the scope of this paper and its page limits.

4. The Cutting Plane Algorithm

We now give an outline of the cutting plane algorithm we have designed for solving the 2-matching problem on complete graphs $K_n = (V, E)$ with weights $c_e \in \mathbb{R}$ for all $e \in E$. As our LP-solver we use IBM's linear programming package MPSX. For this reason the code is written in PL/I and ECL.

(4.1) *Input*: The following data are read: The edge weights c_{ij}, $1 \le i < j \le n$, the parameters NN, D, C (NN is short-hand for "next neighbour" and is used in (4.3) to select the initial set of variables; D and C are parameters that control row elimination and cutting plane recognition, respectively – see (4.7) and (4.8)).

(4.2) *A Heuristic for 2-Matching*: We run a greedy-type algorithm to find a "good" 2-matching M.

(4.3) *Selection of Variables*: We set $E_0 := M$ and, depending on the parameter NN, we add to E_0, for each node v, the NN edges in $\delta(v)$ with smallest weight. E_0 is our initial set of variables. By construction E_0 contains a 2-matching.

(4.4) *Initial LP*: We set up the data structures for the constraints of the initial LP

$$\begin{aligned} x(\delta(v) \cap E_0) &= 2 \quad && \text{for all } v \in V \\ 0 \le x_e &\le 1 && \text{for all } e \in E_0. \end{aligned}$$

We set up a starting basis using M, and define $k := 1$.

(4.5) We call PRIMAL to obtain a minimum solution x^* of the current LP and save the basis.

(4.6) *Optimality Check and Addition of Variables*: If x^* is integral, x^* provides an optimum solution of the 2-matching problem – see (3.2) – on the current set of edges E_0. For every edge $e \in E \setminus E_0$ we generate the corresponding column of the current LP (this has to be implemented very carefully) and calculate the corresponding reduced cost. We also set $k := 1$.

If all the reduced costs have the correct sign, x^* is globally optimal and thus a minimum weight perfect 2-matching for the initial problem. In this case we stop.

Otherwise we add edges to E_0 that may lead to an improvement. We never add more than $n + 500$ edges to E_0, but usually, much less edges have reduced costs with the wrong sign. We call REVISE, restore the old basis and go to (4.5).

(4.7) *Elimination of Cuts*: In this case x^* is not integral. If $D = 1$ and $k = 5$ we eliminate all 2-matching constraints from the current LP that are nonbinding at the current optimum solution x^* and set $k := 1$. (So, using strategy $D = 1$, about every fifth time we reduce the size of our LP. We have tested a few values between 1 and 10; the number 5 seemed to be the best compromise; but it is still quite arbitrary. If $D = 0$ inequalities will never be eliminated.) Otherwise we set $k := k + 1$.

(4.8) *Cutting Plane Recognition*: Since the current optimum solution x^* is nonintegral, a 2-matching constraint violated by x^* does exist.

(a) We call heuristic (3.3). If it finds at least one violated 2-matching constraint we go to (4.9), otherwise we continue with (b).

(b) We run the modified version of the Padberg-Rao procedure described in Section 3.

If $C = 0$, we immediately stop growing the Gomory-Hu tree as soon as a tree edge of weight less than one is found that separates the current Gomory-Hu tree into two odd components. This edge provides one 2-matching constraint violated by x^*, and we go to (4.9).

If $C = 1$ we grow the full Gomory-Hu tree and take all cutting planes that can be read from the final tree. (We have analysed the results of several runs to see whether it pays to check whether the inequalities found define facets of $Q(K_n)$ – see Theorem (2.7) – and to add only those that define facets or to modify the found ones so that the resulting inequalities that are obtained from the Gomory-Hu tree define facets. We observed that most of the inequalities found do indeed define facets and that the additional searching and modifying to get facets only is a waste of time. So we simply add, whatever is found.)

(4.9) *Addition of Inequalities*: The algorithm in (4.8) (a) and (b) provide us with sets $(W_1, T_1), \ldots, (W_p, T_p)$ such that $W_i \subseteq V$, $T_i \subseteq \delta(W_i)$, $|T_i|$ odd and the inequality of type (2.3) (resp. (2.4) and (2.5)) corresponding to W_i and T_i, $i = 1, \ldots, p$ is violated by x^*. For $i = 1, \ldots, p$, we are free to add any of the following three inequalities:

(i) $x(T_i) - x(\delta(W_i) \setminus T_i) \leq |T_i| - 1,$

(ii) $x(E(W_i)) + x(T) \quad \leq |W| + \dfrac{|T| - 1}{2}\,,$

(iii) $x(E(V \setminus W_i)) + x(T) \leq |V \setminus W_i| + \frac{|T|-1}{2}$.

To maintain sparsity one should add the one with the smallest number of nonzeros. We have found empirically that frequently the one of the two inequalities (ii) or (iii) with the smaller set W_i resp. $V \setminus W_i$ of nodes is best in this respect. So we have decided to use only 2-matching constraints of type (2.4). The restriction to one type of inequalities has the additional advantage that the data structures for generating new columns in (4.6) become easier and thus column generation (needed in (4.6)) more efficient.

(4.10) We call REVISE to add the inequalities selected in the way described above, restore the old basis, call DUAL and go to (4.5). □

This finishes the description of our cutting plane algorithm.

Note that the performance of the procedure depends on the choice of the parameters NN, D, and C. It will follow from the computational results reported in Section 5 that, on the average, $5 \leq NN \leq 10$, $D = 1$ and $C = 1$ is the best choice.

2-Matching problems on graphs $G = (V, E)$ that are not complete can be solved by adding all edges not in the graph with large weight. However, it is much better to adjust the selection of initial edges E_0 by setting $E_0 := E$, if G is sparse, or choosing in (4.3) only certain edges that are in G. (In such a case, PRIMAL may report in (4.5) that the current LP is infeasible which proves that the current graph has no perfect 2-matching. Then either further edges of G have to be added or G has no 2-matching.) Moreover, the optimality check in (4.6) through reduced cost calculation should be restricted to the edges in G. It is trivial to implement these changes.

5. Computational Results

We will now demonstrate the influence of the various still open choices of our cutting plane algorithm on the total running time of the code. This way we have determined empirically a procedure which – in our opinion – performes quite reasonably. We have solved 2-matching problems with up to 1000 nodes and thus 499500 variables. The best version of our code handles such problems in less than 1 hour CPU-time on an IBM 4361 Model 5. Of course, since our code has no guaranteed polynomia lupper bound we cannot be sure that this good behaviour will show up always, there maybe runaways in some cases; but due to our computational experience so far we believe that this will not be too frequent. Moreover, we do not know of any further 2-matching code that would be able at all to solve problems of similar size.

We discuss two types of test problems; a few randomly generated problems and some real-world travelling salesman problems. Some of these travelling salesman problems have appeared in the literature – see for instance [1]. We identify a problem by the number of its "cities" and, if this is ambiguous, by an additional letter. We report on a total of 22 different real-world instances named

17, 21, 24, 42, 48, 48 H, 57, 70,

96, 100 A, 100 B, 100 C, 100 D, 100 E, 120,

137, 202, 229, 318, 431, 442, 666.

Moreover, we report here on further 10 random problems

$$100\,R,\ 200\,R,\ \ldots,\ 1000\,R,$$

where "R" stands for "random". The edge weights c_{ij}, $1 \le i < j \le 100, \ldots, 1000$, for these problems have been selected from the set of integers $\{1, 2, \ldots, 5000\}$ randomly and independently.

Table 1 shows the performance of the code for various choices of NN, D, C on the real-world TSP-problems. For each problem, with a few exceptions, we have run 12 different versions of the code that are obtained by setting

$$NN = 5,\ 10,\ \text{or}\ 15;\quad D = 0\ \text{or}\ 1;\quad C = 0\ \text{or}\ 1.$$

In Table 1, the total running time is, for each case, recorded in $\frac{1}{100}$-th of a second. The time includes input, output etc. under VM on the IBM 4361, but does not include system overhead. (The system overhead heavily depends on additional users of the machine.)

Table 1 (and further runs) indicate clearly that, in the large majority of cases, $C = D = 1$ is the best choice of the parameters C and D. This means that the full Gomory-Hu tree should be grown in the Padberg-Rao procedure and that cut elimination should be performed as outlined in (4.7). $NN = 15$ is obviously inferior to the other two choices of NN. In some cases $NN = 5$ is better than $NN = 10$, but there are a number of exceptions where $NN = 10$ shows superior results. The picture here is not clear. As can be seen from Table 5, for the random problems, the choice $NN = 5$ dominates $NN = 10$ by a significant margin.

Based on this, we think that any choice for NN between 5 and 10 is reasonable for the range of problems in question. There are a few "–" in Table 1. In these cases MPSX failed to solve some linear programs that have been generated in the cutting plane algorithm. Due to poor error messages we were unable to recover the reasons for failure. We believe that the breakdown was due to a combination of storage and numerical problems.

For fixed choices $C = D = 1$, and NN in the range 5, 10, 15, Table 2 shows further characteristics of our code that indicate the success of our choice of initial variables, the cutting plane heuristic, etc. The first column of Table 2, named NUM, identifies the problem, the second gives the value for NN, the third column, labelled VALUE, contains the optimum objective function value. The further columns have to be read as follows:

Column 4 $\hat{=}$ VAR:	total number of variables in the last LP solved,
Column 5 $\hat{=}$ ROWS:	the number of equations plus the total number of 2-matching constraints of the final LP,
Column 6 $\hat{=}$ CUTS:	total number of cutting planes generated at all,
Column 7 $\hat{=}$ HC:	number of cutting planes found in heuristic (3.3).

Table 1

NUM	*NN*	*T* *CD*:00	*T* *CD*:01	*T* *CD*:10	*T* *CD*:11
17	5	347	346	362	347
17	10	389	384	406	387
17	15	410	408	421	408
21	5	364	363	377	365
21	10	416	411	421	413
21	15	463	460	471	458
24	5	433	430	437	433
24	10	509	505	511	509
24	15	565	574	580	567
42	5	1016	1031	1048	1018
42	10	1075	1100	1121	1098
42	15	1324	1356	1388	1344
48	5	954	983	1002	1017
48	10	1206	1245	1283	1288
48	15	1413	1447	1272	1260
48*H*	5	855	832	826	824
48*H*	10	683	662	664	660
48*H*	15	829	800	790	789
57	5	1987	1996	1953	1980
57	10	1558	1527	1508	1514
57	15	1945	1931	1906	1921
70	5	4659	4613	4487	4428
70	10	1140	1127	1117	1118
70	15	1503	1485	1464	1463
96	5	3522	3591	3260	3228
96	10	5171	5087	4675	4594
96	15	5067	5141	4267	3916
100*A*	5	2395	2429	2403	2432
100*A*	10	2460	2462	2456	2469
100*A*	15	3118	3131	3116	3133
100*B*	5	16168	16239	14800	13081
100*B*	10	9908	9877	9329	9647
100*B*	15	13163	13236	12217	12635
100*C*	5	5628	5857	3714	3919
100*C*	10	5935	6126	3470	3530
100*C*	15	7824	8004	4499	4511
100*D*	5	5962	5963	5304	5381
100*D*	10	5926	6030	6015	6164
100*D*	15	7898	7969	7999	8105
100*E*	5	2162	2186	2283	2302
100*E*	10	2275	2313	2215	2270
100*E*	15	2836	2872	2759	2811
120	5	13981	15687	14802	12500
120	10	5401	5576	4815	5130
120	15	7227	7055	6680	6841
137	5	4467	4492	4508	4470
137	10	3394	3407	3312	3292
137	15	4819	4835	4604	4607
202	5	8068	7967	8547	8322
202	10	11333	11253	11759	11666
202	15	12537	14104	12553	14145
229	5	59622	41749	53254	34159
229	10	21268	21565	22717	19869
229	15	27321	27822	29030	26017
318	5	538532	310265	198551	109255
318	10	185830	208834	217545	102785
318	15	–	–	–	–
431	5	116592	77728	79636	67803
431	10	185190	142148	104095	68633
431	15	272319	119236	96892	69626
442	5	490336	181122	249078	130985
442	10	360126	305661	109311	88826
442	15	–	333949	193173	171553
666	5	–	487178	735384	399311
666	10	–	1687300	412188	286935
666	15	–	1026072	511314	360513

Table 2

NUM	*NN*	*VALUE*	*VAR*	*ROWS*	*CUTS*	*HC*	*LP*	*IT*	*MFC*
17	5	1684	58	17	0	0	1	1	0
17	10	1684	107	17	0	0	1	1	0
17	15	1684	135	17	0	0	1	1	0
21	5	2707	71	21	0	0	1	1	0
21	10	2707	129	21	0	0	1	1	0
21	15	2707	184	21	0	0	1	1	0
24	5	1227	82	26	2	2	2	1	0
24	10	1227	152	26	2	2	2	1	0
24	15	1227	217	26	2	2	2	1	0
42	5	646	135	50	8	8	6	2	0
42	10	646	261	54	12	12	7	1	0
42	15	646	391	54	12	12	7	1	0
48	5	4805	152	65	18	10	7	1	27
48	10	4805	290	65	18	10	7	1	27
48	15	4805	433	60	12	6	5	1	23
48*H*	5	11197	156	48	0	0	2	2	0
48*H*	10	11197	302	48	0	0	1	1	0
48*H*	15	11197	460	48	0	0	1	1	0
57	5	12679	185	76	19	19	17	2	0
57	10	12679	371	71	14	14	9	1	0
57	15	12679	561	71	14	14	9	1	0
70	5	607	220	109	58	41	32	2	130
70	10	607	415	76	6	6	3	1	0
70	15	607	608	78	8	8	4	1	0
96	5	53069	309	135	44	17	15	2	98
96	10	53069	576	149	55	19	17	2	111
96	15	53069	856	121	30	22	12	1	27
100*A*	5	19564	309	119	19	18	11	2	9
100*A*	10	19564	591	119	19	18	9	1	9
100*A*	15	19564	875	119	19	18	9	1	9
100*B*	5	20664	331	171	178	80	64	2	494
100*B*	10	20664	587	193	132	48	34	1	465
100*B*	15	20664	879	193	132	48	34	1	465
100*C*	5	19861	304	139	50	34	22	2	54
100*C*	10	19861	592	145	48	22	13	1	109
100*C*	15	19861	871	145	48	22	13	1	109
100*D*	5	20269	323	154	65	58	29	2	49
100*D*	10	20269	582	148	66	59	29	1	49
100*D*	15	20269	886	148	66	59	29	1	49
100*E*	5	20752	315	123	25	16	8	2	31
100*E*	10	20752	584	121	22	12	7	1	27
100*E*	15	20752	871	121	22	12	7	1	27
120	5	6694	383	220	166	49	46	2	838
120	10	6694	718	151	40	28	19	1	82
120	15	6694	1069	151	40	28	19	1	82
137	5	67009	428	176	42	36	11	3	25
137	10	67009	813	161	24	18	9	1	25
137	15	67009	1223	161	24	18	9	1	25
202	5	38576	643	258	65	48	20	2	86
202	10	38576	1266	258	65	48	20	2	86
202	15	38576	1891	260	67	50	20	1	86
229	5	128353	727	343	239	118	79	3	481
229	10	128353	1398	315	122	99	36	1	124
229	15	128353	2108	315	122	99	36	1	124
431	5	163905	1365	617	251	166	58	2	384
318	5	39266	1065	507	740	228	105	4	3846
318	10	39266	2008	521	549	184	76	2	2919
318	15	—							
431	10	163905	2653	583	263	134	42	2	551
431	15	163905	3994	583	263	134	41	1	551
442	5	5029	1355	609	432	232	158	3	1634
442	10	5029	2550	579	200	124	82	2	658
442	15	5029	3737	559	336	193	127	1	1661
666	5	286428	2127	1048	797	271	132	3	2237
666	10	286428	4062	980	634	224	151	1	2242
666	15	286428	6090	973	633	225	145	1	2044

Column 8 $\hat{=}$ LP:	number of calls of PRIMAL.
Column 9 $\hat{=}$ IT:	number of times (4.6) has been executed.
Column 10 $\hat{=}$ MFC:	total number of max-flow calculations.

Table 2 is self-explaining. It shows that very few cutting planes and very few variables suffice to prove optimality for the whole problem. In fact, looking at, for instance, the problems 442 and 666 with strategy $NN = 5$, $C = D = 1$, we see that, for problem 442, 1355 of 97461 variables and 609 out of many more than 2^{441} inequalities, and for problem 666, 2127 of 221445 variables and 1048 out of many more than 2^{665} inequalities were enough to produce a globally minimum 2-matching. Table 2 also indicates that our heuristic finds many of the cutting planes and that the number of reiterations (additions of variables and new starts) is very small. The results recorded in Table 2 justify our choices of heuristics, parameters etc. in the design of the cutting plane algorithm.

The distribution of the running time is analysed in Table 3 for the same set of problems and the same choices of parameters as in Table 2. The total running time T is recorded in the third column of Table 3 in $\frac{1}{100}$-th of a second. All further columns of Table 3 contain percentages of this total running time T. The column INP gives the percentage of T needed to read the input; column HEU records the percentage of T spent in executing (4.2) and (4.3); column CON gives the percentage needed for (4.4); and column LP contains the percentage of T spent in calls of PRIMAL, DUAL, and RESTORE (this is essentially the time share needed to solve the linear programs). Column R records the percentage of T spent in constructing the graph G_y, running the cutting plane heuristic, the Padberg-Rao procedure and handling the internal data structures that are needed for adding new cutting planes and doing all kinds of bookkeeping. Column HR contains the share of the time spent in the cutting plane heuristic (4.8) (a); column UP gives the percentage of T needed to prepare data structures for REVISE, to add or delete rows, and to execute REVISE itself. Column OPT records the share of T that was necessary to generate the columns and calculate the reduced costs in (4.6); the percentage of T needed to add new variables through REVISE is given in column MO.

The percentages do not add up to 100%. The remaining part of the time was spent in loading the program, moving control output on the screen etc.

Table 3 shows that the time spent in cutting plane recognition (column R) is marginal, compared to the enormous share used by REVISE and LP-solving. It seems to us that by setting up more efficient revise utilities than MPSX offers, significant running time improvements can be obtained. But this probably makes a complete redesign of such a package necessary. For the 10 random problems, the running time characteristics are displayed in Tables 4 and 5. Table 4 is build up in the same way as Table 2, Table 5 in the same way as Table 3.

Tables 4 and 5 show that random problems of the type described above are much easier for our code than real-world problems. Optimality is proved with astonishingly few cutting planes and variables. For instance, in problem 1000 R with $NN = 5$ less than 0.7% of the variables were involved and only two 2-matching constraints.

Table 3

NUM	*NN*	*T*	*INP*	*HEU*	*CON*	*LP*	*R*	*HR*	*UP*	*OPT*	*MO*
17	5	347	23.05	15.27	23.34	17.29	0.58	0.00	0.58	9.22	0.00
17	10	387	20.67	17.57	24.29	17.31	0.52	0.00	0.52	8.53	0.00
17	15	408	19.61	18.87	24.75	16.91	0.49	0.00	0.49	8.33	0.00
21	5	365	22.19	16.44	22.47	18.08	0.55	0.00	0.55	9.04	0.00
21	10	413	19.61	18.89	23.97	17.19	0.73	0.00	0.24	8.47	0.00
21	15	458	18.34	20.74	25.11	16.59	0.44	0.00	0.44	7.86	0.00
24	5	433	18.94	14.09	19.86	20.79	1.15	0.00	7.39	8.31	0.00
24	10	509	16.70	16.70	20.83	20.63	0.98	0.20	7.47	7.47	0.00
24	15	567	15.17	18.17	22.05	19.93	0.88	0.18	7.76	7.05	0.00
42	5	1018	10.12	8.15	10.22	24.07	2.75	0.29	11.49	8.55	14.64
42	10	1098	8.56	11.20	13.11	29.60	3.19	0.55	23.41	4.83	0.00
42	15	1344	7.29	12.87	13.62	29.39	2.60	0.30	24.63	4.17	0.00
48	5	1017	9.83	9.24	10.91	28.71	5.01	0.59	24.29	5.90	0.00
48	10	1288	8.07	10.79	11.88	28.34	4.11	0.47	26.09	5.05	0.00
48	15	1260	8.57	15.16	15.71	27.70	3.02	0.32	19.21	4.52	0.00
48*H*	5	824	13.35	11.89	13.47	20.63	1.21	0.00	0.36	10.07	17.72
48*H*	10	660	15.15	21.21	23.64	23.03	0.91	0.00	0.15	7.27	0.00
48*H*	15	789	12.93	24.33	25.73	20.91	0.76	0.00	0.25	6.84	0.00
57	5	1980	5.91	5.25	5.86	29.24	5.10	0.76	26.77	6.46	8.79
57	10	1514	7.07	10.63	11.16	30.52	3.50	0.59	27.61	4.16	0.00
57	15	1921	5.73	11.61	11.97	29.67	2.86	0.47	29.46	3.59	0.00
70	5	4428	2.94	2.71	2.87	33.90	6.91	0.93	35.73	5.56	5.22
70	10	1118	10.55	16.64	16.55	30.77	2.06	0.18	11.18	5.81	0.00
70	15	1463	8.27	17.36	16.95	27.68	2.19	0.21	15.93	5.13	0.00
96	5	3228	5.58	4.89	5.05	32.03	6.72	0.59	23.67	9.79	6.97
96	10	4594	4.14	5.57	5.44	30.19	5.68	0.41	27.91	8.62	7.58
96	15	3916	4.39	9.04	8.48	35.42	3.63	0.46	31.33	3.96	0.00
100*A*	5	2432	7.85	6.99	6.78	29.73	5.18	0.45	18.09	10.20	8.84
100*A*	10	2469	6.93	10.65	10.41	32.77	4.17	0.53	25.11	5.35	0.00
100*A*	15	3133	5.65	11.65	10.85	32.91	3.42	0.35	26.78	4.50	0.00
100*B*	5	13081	1.43	1.34	1.31	35.45	7.58	0.73	39.57	5.99	4.62
100*B*	10	9647	1.79	2.76	2.65	34.41	7.05	0.55	45.00	4.18	0.00
100*B*	15	12635	1.41	2.93	2.71	33.72	5.48	0.43	48.82	3.04	0.00
100*C*	5	3919	4.70	4.31	4.21	32.92	6.61	0.74	28.86	8.22	5.41
100*C*	10	3530	4.84	7.59	7.37	30.82	5.89	0.48	33.71	6.12	0.00
100*C*	15	4511	3.90	8.07	7.56	31.97	4.70	0.38	35.54	4.81	0.00
100*D*	5	5381	3.48	3.23	3.10	34.01	6.24	0.59	30.94	8.85	5.89
100*D*	10	6164	2.79	4.27	4.10	36.26	5.48	0.60	40.87	3.67	0.00
100*D*	15	8105	2.18	4.54	4.17	36.18	4.18	0.53	43.48	2.83	0.00
100*E*	5	2302	8.08	7.47	7.25	27.67	4.60	0.30	15.42	12.42	10.69
100*E*	10	2270	7.62	11.54	11.15	32.29	4.14	0.35	22.11	6.34	0.00
100*E*	15	2811	6.30	12.98	11.99	31.91	3.27	0.36	23.41	5.37	0.00
120	5	12500	1.68	1.62	1.51	32.66	9.29	0.62	35.88	9.56	4.83
120	10	5130	3.76	6.16	6.10	35.13	5.83	0.51	35.65	4.46	0.00
120	15	6841	3.00	6.65	6.11	34.51	4.43	0.42	39.15	3.44	0.01
137	5	4470	6.33	5.06	4.54	25.77	3.96	0.29	13.58	21.05	12.86
137	10	3292	7.44	10.63	10.27	32.56	4.56	0.39	23.06	7.20	0.00
137	15	4607	5.90	11.92	11.46	32.17	3.56	0.33	26.29	5.43	0.00
202	5	8322	5.37	4.12	3.42	31.98	5.71	0.43	20.98	20.56	4.16
202	10	11666	4.03	4.78	4.10	33.75	4.23	0.35	26.55	14.77	4.06
202	15	14145	3.12	5.44	4.79	42.20	3.53	0.32	32.66	6.23	0.00
229	5	34159	1.72	1.15	0.88	33.83	6.44	0.64	31.75	19.04	3.26
229	10	19869	2.67	3.21	2.58	40.84	4.77	0.44	36.44	8.07	0.00
229	15	26017	2.06	3.41	2.78	40.05	3.74	0.36	40.40	6.21	0.00
318	5	109255	0.93	0.56	0.38	31.42	6.34	0.39	28.68	26.18	4.27
318	10	102785	0.92	0.93	0.69	36.24	4.95	0.31	39.74	14.06	1.52
318	15	—									
431	5	67803	2.43	1.34	0.81	26.49	4.28	0.38	19.31	41.69	2.52
431	10	68633	2.49	1.97	1.38	27.67	3.41	0.26	26.55	33.82	1.42
431	15	69626	2.36	2.65	1.96	33.73	3.39	0.26	38.48	16.49	0.00
442	5	130985	1.34	0.72	0.40	34.04	6.57	0.54	31.76	22.55	1.48
442	10	88826	1.96	1.59	0.99	33.44	4.98	0.39	34.62	19.85	1.40
442	15	171553	0.99	1.11	0.72	34.98	4.48	0.33	52.68	4.44	0.00
666	5	399311	0.95	0.43	0.21	18.72	2.88	0.21	15.13	59.45	1.66
666	10	286935	1.26	0.83	0.50	34.35	4.67	0.32	36.07	21.76	0.00
666	15	360513	1.01	0.87	0.58	34.47	3.62	0.25	42.23	16.75	0.00

Table 4

NUM	*NN*	*VALUE*	*VAR*	*ROWS*	*CUTS*	*HC*	*LP*	*IT*	*MFC*
100*R*	5	9639	315	100	0	0	1	1	0
100*R*	10	9639	574	100	0	0	1	1	0
200*R*	5	9554	647	208	8	8	6	2	0
200*R*	10	9554	1182	221	22	8	7	1	144
300*R*	5	10283	1000	300	0	0	3	3	0
300*R*	10	10283	1789	300	0	0	1	1	0
400*R*	5	9494	1310	404	4	4	6	4	0
400*R*	10	9494	2392	402	2	2	2	1	0
500*R*	5	9166	1614	502	2	2	4	3	0
500*R*	10	9166	2952	502	2	2	2	1	0
600*R*	5	9579	1947	602	2	2	3	2	0
600*R*	10	9579	3558	602	2	2	2	1	0
700*R*	5	10129	2301	737	41	17	13	2	258
700*R*	10	10129	4148	774	109	12	20	1	1712
800*R*	5	10100	2631	800	0	0	3	3	0
800*R*	10	10100	4743	800	0	0	1	1	0
900*R*	5	10003	2936	905	5	4	6	3	97
900*R*	10	10003	5331	900	0	0	1	1	0
1000*R*	5	9970	3280	1002	2	2	4	3	0
1000*R*	10	9970	5913	1002	2	2	2	1	0

Table 5

NUM	*NN*	*S*/100	*INP*	*HEU*	*CON*	*LP*	*R*	*HR*	*UP*	*OPT*	*MO*
100*R*	5	1116	14.70	15.68	15.05	41.58	0.90	0.00	0.18	6.09	0.00
100*R*	10	1452	11.71	17.98	17.36	40.77	0.76	0.00	0.14	5.30	0.00
200*R*	5	4867	9.00	7.17	5.65	47.26	2.49	0.21	7.73	10.07	5.79
200*R*	10	6926	6.04	7.88	6.25	52.32	3.58	0.25	15.19	6.12	0.00
300*R*	5	10849	8.26	5.37	3.80	59.19	0.86	0.00	0.06	7.38	11.29
300*R*	10	12930	6.44	6.86	5.07	77.56	0.25	0.00	0.02	2.27	0.00
400*R*	5	17498	9.00	4.84	3.03	51.62	1.41	0.05	1.90	13.62	10.44
400*R*	10	16153	8.96	7.87	5.47	70.43	0.51	0.02	1.62	3.47	0.00
500*R*	5	29990	7.37	3.90	2.19	69.57	0.68	0.01	0.66	7.25	6.23
500*R*	10	38079	5.56	4.46	2.86	83.05	0.27	0.01	0.80	2.14	0.00
600*R*	5	38717	7.92	3.91	2.01	76.12	0.47	0.02	0.58	5.57	2.06
600*R*	10	118807	2.55	1.83	1.08	92.88	0.10	0.00	0.28	0.95	0.00
700*R*	5	71773	5.74	2.69	1.23	65.73	1.76	0.11	7.20	11.54	3.07
700*R*	10	127905	3.18	2.09	1.13	67.68	3.10	0.12	12.42	9.88	0.00
800*R*	5	63642	8.56	3.70	1.64	69.65	0.39	0.00	0.01	7.03	7.29
800*R*	10	104286	5.11	3.11	1.66	87.98	0.08	0.00	0.00	1.54	0.00
900*R*	5	120045	5.80	2.42	1.01	72.53	0.59	0.02	1.01	7.83	7.73
900*R*	10	172660	3.97	2.25	1.17	91.03	0.06	0.00	0.00	1.15	0.00
1000*R*	5	134142	6.34	2.55	0.98	71.31	0.30	0.01	0.25	6.42	10.77
1000*R*	10	437568	1.88	1.02	0.50	95.59	0.05	0.00	0.12	0.68	0.00

Thus, almost all of the time is spent in solving the initial linear program set up in (4.4).

Since there are no other 2-matching codes available that work for the same problem range as our algorithm, we could not compare the performance of our code with others. Our code for the 1-matching problem described in [7], however, belongs to the best 1-matching codes available. By running this 1-matching code on the same machine and the same problems we found that, for random problems, the 2-matching code needs not more 20% – 30% more time. For the real-world problems there is no clear picture. Sometimes the running times for the 2-matching problems are shorter (not too often), sometimes comparable, more frequently about 50% slower, but occasionally they are 3 times as long as the times needed to find a shortest perfect 1-matching on the same data set. The 2-matching problem is generally considered to be – in a sense – harder than the 1-matching problem. This is in some way reflected by these comparisons. But it seems to us that there is no way to solve 2-matching problems as fast as we can do it with our code by reducing them to 1-matching problems.

References

[1] Crowder, H., Padberg, M. W.: Solving large-scale symmetric travelling salesman problems. Management Science *26*, 495–509 (1980).

[2] Edmonds, J.: Maximum matching and a polyhedron with 0,1-vertices. Journal of Research of the National Bureau of Standards – *B69B*, 125–130 (1965).

[3] Edmonds, J., Johnson, E. L., Lockhart, S.: Blossom I, a code for matching. Internal IBM Report. Yorktown Heights: T. J. Watson Research Center 1969.

[4] Glover, F., Klingman, D., Mote, J., Whitman, D.: A primal simplex variant for the maximum flow problem. Center for Cybernetics Studies. Preprint CCS 362, Austin, TX, 1979.

[5] Gomory, R. E., Hu, T. C.: Multi terminal network flows. J. Soc. Industr. Applied Math. *9*, 551–571 (1961).

[6] Grötschel, M.: Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme. Meisenheim: Hain 1977.

[7] Grötschel, M., Holland, O.: Solving matching problems with linear programming. Mathematical Programming *33*, 243–259 (1985).

[8] Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica *1*, 169–197 (1981).

[9] Lovász, L., Plummer, M. D.: Matching Theory. Budapest: Akadémiai Kiadó 1986.

[10] Padberg, M. W., Rao, M. R.: Odd minimum cut-sets and b-matchings. Mathematics of Operations Research *7*, 67–80 (1982).

Martin Grötschel
Lehrstuhl für Angewandte Mathematik II
Universität Augsburg
Memminger Strasse 6
D-8900 Augsburg
German Federal Republic

Olaf Holland
Institut für Operations Research
Universität Bonn
Nassestrasse 2
D-5300 Bonn 1
German Federal Republic